Оглавление

Оставшаяся часть приложения уже вам знакома. Если снова запустить приложение, вы увидите бесконечную отправку сообщений. Взгляните во время его работы на консоль RabbitMQ. Можно также вставить цикл `for` для отправки большего числа сообщений за полсекунды.

Удаленный сервер RabbitMQ

Для доступа к удаленному серверу RabbitMQ необходимо добавить в файл `application.properties` следующие свойства[1]:

```
spring.rabbitmq.host=mydomain.com
spring.rabbitmq.username=rabbituser
spring.rabbitmq.password=thisissecured
spring.rabbitmq.port=5672
spring.rabbitmq.virtual-host=/production
```

Вы всегда можете узнать обо всех свойствах RabbitMQ в документации Spring Boot по адресу https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html.

Теперь вы знаете, насколько легко использовать RabbitMQ со Spring Boot. Если хотите узнать больше о RabbitMQ и технологии Spring AMQP, то можете получить дополнительную информацию на основной странице этого проекта: http://projects.spring.io/spring-amqp/.

Можете теперь остановить сервер RabbitMQ, нажав Ctrl+C в терминале, где запустили его. Существуют и другие возможности использования RabbitMQ, например создание кластера и обеспечение высокой доступности. Больше информации об этом можно найти на сайте www.rabbitmq.com.

Обмен сообщениями в Redis с помощью Spring Boot

Настала очередь Redis. Redis (REmote DIctionary Server) представляет собой базу данных NoSQL — хранилище пар «ключ/значение». Redis написана на языке C и, несмотря на небольшой объем используемой оперативной памяти, отличается высокой надежностью, масштабируемостью, быстродействием и большими возможностями. Основная ее задача — хранение структур данных,

[1] Разумеется, поменяв соответствующим образом их значения.

таких как списки, хеши, строковые значения, множества и отсортированные множества. Одна из основных ее возможностей — обмен сообщениями по типу «публикация/подписка». Поэтому мы и воспользуемся Redis в качестве брокера сообщений.

Установка Redis

Процедура установки Redis очень проста. Если вы используете Mac OS X/Linux, можете воспользоваться командой `brew` и выполнить следующее:

```
$ brew update && brew install redis
```

Если же вы работаете в Unix или Windows-системе, то можете зайти на вебсайт Redis и скачать один из установочных пакетов Redis[1] по адресу http://redis.io/download.

Создание приложения ToDo с использованием Redis

Использовать Redis для обмена сообщениями по типу «публикация/подписка» очень просто, никаких серьезных отличий от других технологий нет. Мы сейчас реализуем отправку и получение запланированных дел на основе паттерна публикации/подписки с помощью Redis.

Для начала откройте свой любимый браузер и перейдите на сайт Spring Initializr. Добавьте следующие значения в соответствующие поля.

- Group (Группа): `com.apress.todo`.
- Artifact (Артефакт): `todo-redis`.
- Name (Название): `todo-redis`.
- Package Name (Название пакета): `com.apress.todo`.
- Dependencies (Зависимости): `Redis`, `Web`, `Lombok`, `JPA`, `REST Repositories`, `H2`, `MySQL`.

Можете выбрать в качестве типа проекта Maven или Gradle. Затем нажмите кнопку Generate Project (Сгенерировать проект) и скачайте ZIP-файл. Разархивируйте его и импортируйте в вашу любимую IDE (рис. 9.6).

---

[1] Уточнение: на этом сайте доступен только исходный код, который необходимо скомпилировать для получения исполняемых файлов. Впрочем, существуют и бинарные сборки, правда, для Windows они все довольно сильно отстают от текущей версии.

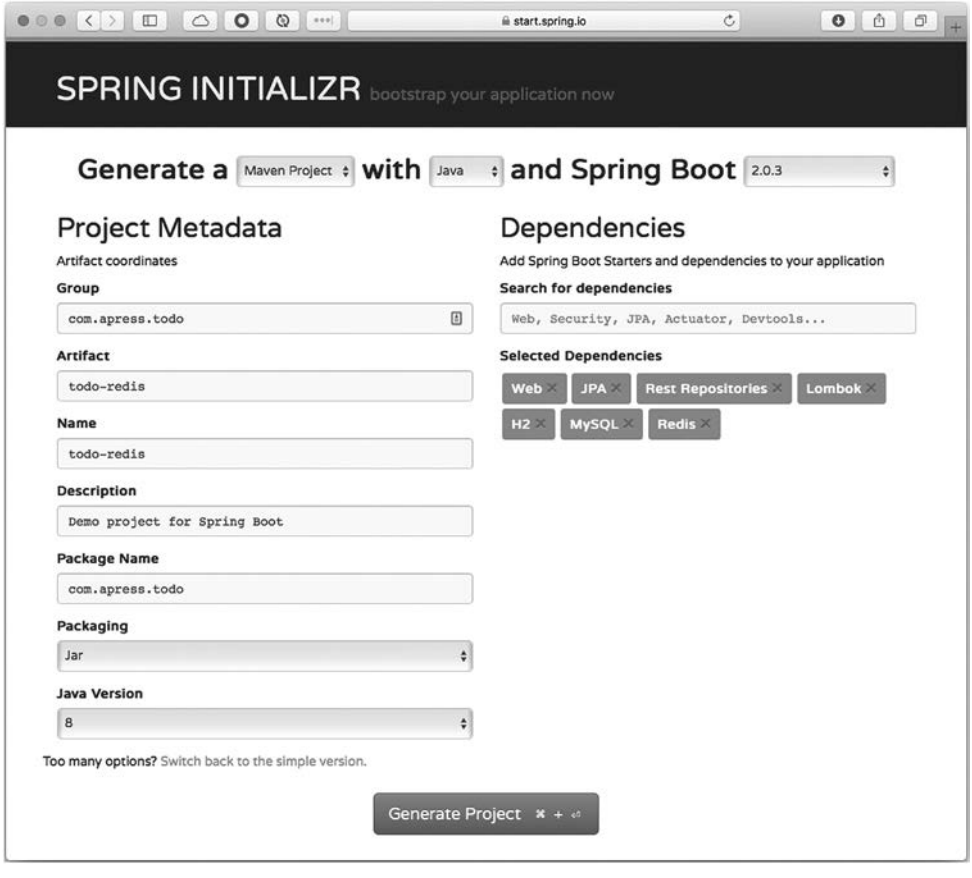

Рис. 9.6. Spring Initializr

Мы воспользуемся классом предметной области ToDo и репозиторием из предыдущих глав.

Генератор ToDo

Начнем с создания класса генератора, задача которого будет состоять в отправке экземпляра `ToDo` в конкретную тему (листинг 9.12).

Листинг 9.12. com.apress.todo.redis.ToDoProducer.java

```
package com.apress.todo.redis;

import com.apress.todo.domain.ToDo;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;

@Component
public class ToDoProducer {

    private static final Logger log = LoggerFactory.getLogger(ToDoProducer.
        class);
    private RedisTemplate redisTemplate;

    public ToDoProducer(RedisTemplate redisTemplate){
        this.redisTemplate = redisTemplate;
    }

    public void sendTo(String topic, ToDo toDo){
        log.info("Producer> ToDo sent");
        this.redisTemplate.convertAndSend(topic, toDo);
    }
}
```

В листинге 9.12 приведен класс генератора. Он практически идентичен предыдущим технологиям. В нем используется класс паттерна `*Template`; в данном случае `RedisTemplate`, отправляющий экземпляры `ToDo` в конкретную тему.

Потребитель ToDo

Далее создадим потребитель для подписки на тему (листинг 9.13).

Листинг 9.13. com.apress.todo.redis.ToDoConsumer.java

```
package com.apress.todo.redis;

import com.apress.todo.domain.ToDo;
import com.apress.todo.repository.ToDoRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Component
public class ToDoConsumer {

    private static final Logger log =
        LoggerFactory.getLogger(ToDoConsumer.class);
    private ToDoRepository repository;
```

```
    public ToDoConsumer(ToDoRepository repository){
        this.repository = repository;
    }

    public void handleMessage(ToDo toDo) {
        log.info("Consumer> " + toDo);
        log.info("ToDo created> " + this.repository.save(toDo));
    }
}
```

В листинге 9.13 приведен потребитель, который подписывается на тему для поступающих сообщений `ToDo`. Важно понимать, что для использования прослушивателя обязательно нужен метод с названием `handleMessage` (это ограничение, налагаемое при создании `MessageListenerAdapter`).

Конфигурация приложения ToDo

Теперь создадим конфигурацию для нашего приложения ToDo (листинг 9.14).

Листинг 9.14. com.apress.todo.config.ToDoConfig.java

```
package com.apress.todo.config;

import com.apress.todo.domain.ToDo;
import com.apress.todo.redis.ToDoConsumer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;

@Configuration
public class ToDoConfig {

    @Bean
    public RedisMessageListenerContainer container(RedisConnectionFactory
        connectionFactory,
                                MessageListenerAdapter toDoListenerAdapter,
                                @Value("${todo.redis.topic}") String topic) {
        RedisMessageListenerContainer container = new
            RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
```

```
        container.addMessageListener(toDoListenerAdapter, new
            PatternTopic(topic));
        return container;
    }

    @Bean
    MessageListenerAdapter toDoListenerAdapter(ToDoConsumer consumer) {
        MessageListenerAdapter messageListenerAdapter =
            new MessageListenerAdapter(consumer);
        messageListenerAdapter.setSerializer(new
            Jackson2JsonRedisSerializer<>(ToDo.class));
        return messageListenerAdapter;
    }

    @Bean
    RedisTemplate<String, ToDo> redisTemplate(RedisConnectionFactory
        connectionFactory){
        RedisTemplate<String,ToDo> redisTemplate = new
            RedisTemplate<String,ToDo>();
        redisTemplate.setConnectionFactory(connectionFactory);
        redisTemplate.setDefaultSerializer(new Jackson2JsonRedisSerializer<>(
            ToDo.class));
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }
}
```

В листинге 9.14 приведена необходимая для приложения ToDo конфигурация. В этом классе объявлены следующие компоненты Spring.

- Класс `RedisMessageListenerContainer`. Этот класс отвечает за подключение к теме Redis.
- Класс `MessageListenerAdapter`. Этот адаптер использует класс простого Java-объекта в старом стиле (Plain Old Java Object, POJO) для обработки сообщения. Обязательным требованием является название `handleMessage` для обрабатывающего сообщение метода, который получает сообщение из темы в виде экземпляра `ToDo`, вследствие чего требуется также сериализатор.
- Класс `Jackson2JsonRedisSerializer`. Этот сериализатор производит преобразование из экземпляра `ToDo`/в него.
- Класс `RedisTemplate`. Этот класс реализует паттерн `Template` и аналогичен используемым для прочих технологий обмена сообщениями. Для работы с JSON и преобразованием из экземпляров `ToDo`/в них ему требуется сериализатор.

Эти дополнительные настройки необходимы для работы с форматом JSON и корректного преобразования из экземпляров `ToDo`/в них; но вы можете без них обойтись и воспользоваться конфигурацией по умолчанию, для которой требуется сериализуемый объект (например, типа `String`) для отправки. Вместо него можно применить `StringRedisTemplate`.

Добавьте в файл `application.properties` следующее содержимое:

```
# JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop

# ToDo Redis
todo.redis.topic=todos
```

Запуск приложения ToDo

Перед запуском приложения ToDo убедитесь, что сервер Redis запущен. Для запуска сервера выполните в терминале следующую команду:

```
$ redis-server
89887:C 11 Feb 20:17:55.320 # Warning: no config file specified, using the
default config. In order to specify a config file use redis-server /path/
to/redis.conf
89887:M 11 Feb 20:17:55.321 * Increased maximum number of open files to
10032 (it was originally set to 256).
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 4.0.10 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
      (    '      ,       .-`  | `,    )     Standalone mode
      |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
      |    `-._   `._    /     _.-'    |     PID: 89887
       `-._    `-._  `-./  _.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |     http://redis.io
       `-._    `-._`-.__.-'_.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
           `-._    `-.__.-'    _.-'
               `-._        _.-'
                   `-.__.-'
89887:M 11 Feb 20:17:55.323 # Server started, Redis version 3.0.7
89887:M 11 Feb 20:17:55.323 * The server is now ready to accept connections
on port 6379
```

Этот вывод демонстрирует, что сервер Redis готов к использованию и прослушивает на порте 6379. Откройте новое окно терминала и выполните следующую команду:

```
$ redis-cli
```

Это клиент командной оболочки, который умеет подключаться к серверу Redis. Можете подписаться на тему `todos`, выполнив следующую команду.

```
127.0.0.1:6379> SUBSCRIBE todos
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "todos"
3) (integer) 1
```

Теперь можно запустить приложение как обычно (из IDE или с помощью Maven/Gradle). Если вы используете Maven, выполните:

```
$ ./mvnw spring-boot:run
```

После выполнения этой команды вы должны увидеть в журнале примерно следующее:

```
...
Producer> Message Sent
Consumer> ToDo(id=null, description=workout tomorrow morning!,
 created=null, modified=null, completed=false)
ToDo created> ToDo(id=8a808087645bd67001645bd6785b0000, description=workout
 tomorrow morning!, created=2018-07-02T10:32:19.546, modified=2018-07-
 02T10:32:19.547, completed=false)
...
```

Если теперь заглянуть в командную оболочку Redis, вы должны увидеть там что-то вроде следующего:

```
1) "message"
2) "todos"
3)  "{\"id\":null,\"description\":\"workout tomorrow morning!\",\"created\":
 null,\"modified\":null,\"completed\":false}"
```

И конечно, вы можете увидеть новое запланированное дело в браузере по адресу http://localhost:8080/toDos.

Отлично! Вы создали приложение Spring Boot для обмена сообщениями, использующее Redis. Можете теперь остановить сервер Redis, нажав Ctrl+C.

Удаленный сервер Redis

Для доступа к удаленному серверу Redis необходимо добавить в файл `application.properties` следующие свойства:

```
spring.redis.database=0
spring.redis.host=localhost
spring.redis.password=mysecurepassword
spring.redis.port=6379
```

Вы всегда можете узнать обо всех свойствах Redis в документации Spring Boot по адресу https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html.

Теперь вы знаете, как использовать Redis в качестве брокера сообщений. Если хотите узнать больше о применении хранилищ типа «ключ/значение» со Spring Boot, можете заглянуть на страницу проекта Spring Data Redis: http://projects.spring.io/spring-data-redis/.

Использование WebSockets со Spring Boot

Возможно, логичнее было бы поместить обсуждение WebSockets в главу, посвященную веб-приложениям, но мне кажется, что WebSockets относится скорее к обмену сообщениями, поэтому я поместил данный раздел в эту главу.

WebSockets — новый способ связи, заменяющий веб-технологию «клиент/сервер». Он делает возможным длительные односокетные TCP-соединения между клиентом и сервером. Его также называют технологией *проталкивания* (push), поскольку сервер может отправлять данные без необходимости выполнения клиентом долгих опросов для запроса обновлений.

В этом разделе показан пример, включающий отправку сообщения через конечную точку REST (`Producer`) и получение сообщений (`Consumer`) с помощью веб-страницы и библиотек JavaScript.

Создание приложения ToDo с использованием WebSockets

Создадим приложение ToDo, использующее репозитории JPA REST. Каждое появляющееся новое запланированное дело выводится на веб-странице.